Interval scheduling maximizing minimum coverage

Mäkinen, Veli

2017-07-10

Mäkinen , V , Staneva , V , Tomescu , A I , Valenzuela , D & Wilzbach , S 2017 , ' Interval scheduling maximizing minimum coverage ' , Discrete Applied Mathematics , vol. 225 , pp. 130-135 . https://doi.org/10.1016/j.dam.2016.08.021

http://hdl.handle.net/10138/307575
https://doi.org/10.1016/j.dam.2016.08.021

Interval scheduling maximizing minimum coverage

Veli Mäkinen, Valeria Staneva[1], Alexandru I. Tomescu, Daniel Valenzuela*

Helsinki Institute for Information Technology HIIT
Department of Computer Science
University of Helsinki, Finland

Abstract

In the classical interval scheduling type of problems, a set of n jobs, characterized by their start and end time, need to be executed by a set of machines, under various constraints. In this paper we study a new variant in which the jobs need to be assigned to at most k identical machines, such that the minimum number of machines that are busy at the same time is maximized. This is relevant in the context of genome sequencing and haplotyping, specifically when a set of DNA reads aligned to a genome needs to be pruned so that no more than k reads overlap, while maintaining as much read coverage as possible across the entire genome. We show that the problem can be solved in time $\min\left(O(n^2 \log k/\log n), O(nk \log k)\right)$ by using max-flows. We also give an $O(n \log n)$-time approximation algorithm with approximation ratio $\rho = \frac{k}{\lfloor k/2 \rfloor}$.

Keywords: Interval scheduling, read pruning, haplotype assembly, max-flows

1. Introduction

Interval scheduling is a classical problem in combinatorial optimization. The input usually consists of n jobs, such that each job j needs to be executed in the time interval $[s_j, f_j)$, by any available machine. In the most basic variant of this problem, each machine is always available, can process at most one job at a time, and once it starts executing a job it does so until it is finished. The task is to process all jobs using the minimum number of machines [9]. This is solvable in time $O(n \log n)$ [8]. In another problem variant, known as *interval scheduling with given machines*, there are only k available machines, and the execution of each job brings a specified profit. The task is to schedule a maximum-profit subset of jobs. This is also solvable in polynomial time, for example by min-cost flows [1, 2]. Some problem variants are NP-hard, for example if each job can be executed only by a given subset of machines [1], or if each machine is available during a specific period of time [3]. See the surveys [9, 10] for further references.

Most previous work has focused on either maximizing the profit obtained from executing the jobs, or on minimizing the resources used by the jobs. In this paper we study a new problem variant with a rather different objective function, motivated by a new application of interval scheduling in genome haplotyping with high-throughput DNA sequencing. In this variant, which we call *interval*

*Corresponding author

Email address: dvalenzu@cs.helsinki.fi (Daniel Valenzuela)

[1]Current affiliation MIT – Massachusetts Institute of Technology (student). Work conducted while visiting University of Helsinki as summer intern.

Figure 1: An instance of interval scheduling maximizing minimum coverage in which 8 intervals are given and $k = 2$ machines are available to execute them. Observe that in all solutions to this problem three disjoint intervals of length 5 need to be removed, leading to a solution that executes 5 jobs and the number of idle machines is never greater than 1. However, in the solutions to the classical interval scheduling with given machines problem, the two intervals of length 7 are removed both in the case when all intervals have the same profit, and in the case when the profit of an interval equals its length.

scheduling maximizing minimum coverage, we need to select a subset of jobs to be executed by a given number k of machines, such that the minimum, over the number of machines that are busy at any given time, is as large as possible. Fig. 1 gives an example. To the best of our knowledge this variant has not been addressed before.

The rest of the paper is structured as follows. In Sec. 2 we discuss our original motivation in the context of high-throughput DNA sequencing and give the precise problem formulation. In Sec. 3 we present a reduction to a max-flow problem, which leads to a $O(n^2 \log k / \log n)$ solution for our problem. In Sec. 4 we present a tailored max-flow algorithm that runs in $O(nk \log k)$ time, which is faster than the previous when $k = o(n/\log n)$. Since for large k the best complexity is almost quadratic, we also study a way to find approximate solutions: In Sec. 5 we present an $O(n \log n)$-time $\frac{k}{\lfloor k/2 \rfloor}$-approximation algorithm.

2. Haplotype phasing, read pruning and interval scheduling

High-throughput sequencing is a technique developed over the last decade that can produce millions of DNA fragments, called *reads*, from random positions across the genome of an individual. Depending on the technology, their length can be from hundreds to thousands of characters. Many analyses are carried out by first aligning the reads to a reference genome sequence of the species, and studying, for example, the genetic variations of the individual with respect to the reference (see e.g. [12]). A more detailed analysis, called *haplotype phasing*, also takes into account the fact that in some species, such as humans, each chromosome is present in two copies, inherited from each parent. In this context it is also desirable to assign the genetic variations to the copy of the chromosome where they are present.

Since real data has sequencing and alignment errors, a well-known problem formulation asks for the minimum number of corrections that enables a consistent partitioning of the input set of reads into the two copies of the chromosome they were sequenced from. This problem is called *minimum error correction* and was introduced by Lippert *et al.* in [11] and proved NP-hard in [4]. A practical algorithm for this problem was proposed in [14], having a time complexity of $O(2^{k-1}m)$, where m is the proportional to the length of the genome, and k is the maximum number of reads covering any position of the genome. This algorithm is particularly useful because its runtime is independent of the read length.

The higher the number of reads, and the more uniform they are distributed across the genome, the more accurate the solution to the minimum error correction problem is in practice. However, the $O(2^{k-1}m)$ time complexity makes this algorithm feasible only for small values of k. In its implementation [14], for every genomic position with too high read coverage, some reads are removed

at random. However, this may arbitrarily lead to some other positions having a too low coverage for accurate results. In this paper we study the problem of pruning the read set such that the maximum read coverage is less than a given integer k, and the minimum coverage across all genomic positions is as high as possible.

Our formal definition is as follows. We will represent each read i as an interval $[s_i, f_i)$. We will assume that $0 \leqslant s_i < f_i < N$. Given an interval $[s_i, f_i)$ and a point $p \in [s_i, f_i)$, we say that $[s_i, f_i)$ *covers* p. If $p \in (s_i, f_i)$ we say that $[s_i, f_i)$ *strictly covers* p. Given a set $S = \{[s_i, f_i) \ : \ i \in \{1, \ldots, n\} \mid s_i < f_i\}$ of intervals, and a point p we define the *coverage of* p as $\mathsf{cov}_S(p) = |\{[s_i, f_i) \in S | [s_i, f_i) \text{ covers } p\}|$. When clear from the context, the subscript S will be omitted. We also define the *maximum coverage of* S as $\mathsf{maxcov}(S) = max_{p \in [0,N)} \mathsf{cov}_S(p)$. Likewise, we define the *minimum coverage of* S as $\mathsf{mincov}(S) = min_{p \in [0,N)} \mathsf{cov}_S(p)$. Our problem is the following one.

Problem 1 (Interval scheduling maximizing minimum coverage)**.**
INPUT. *A set* $S = \{[s_i, f_i) \ : \ i \in \{1, \ldots, n\} \mid s_i < f_i\}$ *of intervals and an integer* k*.*
TASK. *Find an* $S' \subseteq S$ *such that* $\mathsf{maxcov}(S') \leqslant k$ *and maximizing* $\mathsf{mincov}(S')$*.*

Note that if we only keep the first condition, namely $S' \subseteq S$ such that $\mathsf{maxcov}(S') \leqslant k$ our problem would be exactly the one of finding a feasible set of jobs to be scheduled on k machines.

3. The reduction to max-flows

In this section we show that the problem is solvable in time $O(n^2 \log k / \log n)$ by max-flows. First, we consider the decision version of the maximization problem, as follows.

Problem 2 (Interval scheduling with bounded coverage)**.**
INPUT. *A set* $S = \{[s_i, f_i) \ : \ i \in \{1, \ldots, n\} \mid s_i < f_i\}$ *of intervals, and integers* k, t*.*
TASK. *Decide if there an* $S' \subseteq S$ *such that* $\mathsf{maxcov}(S') \leqslant k$ *and* $\mathsf{mincov}(S') \geqslant t$*, and if yes, output such* S'*.*

We now describe the reduction of Problem 2 to a max-flow problem, following standard network flow notation, as can be found e.g. in [12]. See also Fig. 2(b) for an example. Let $s = \min_{i \in \{1,\ldots,n\}} s_i$ and $f = \max_{i \in \{1,\ldots,n\}} f_i$. We construct a graph $G_{S,k,t}$ (possibly having parallel arcs) whose vertex set equals $\{s-1, f+1\} \cup \{s_i, f_i \ : \ i \in \{1, \ldots, n\}\}$. Vertex $s-1$ will be the unique source of the graph, and vertex $f+1$ will be the unique sink of the graph. For every two consecutive numbers i, j in $V(G)$ (i.e., such that there is no number p in $V(G)$ with $i < p < j$), we add the arc (i, j) to $E(G)$. We call these arcs *backbone* arcs. The backbone arcs $(s-1, s)$ and $(f, f+1)$ receive capacity k, and the other backbone arcs have capacity $k - t$. For every interval $[s_i, f_i) \in S$, we add to $E(G)$ the arc (s_i, f_i) with capacity 1. We call these latter arcs *interval* arcs.

Theorem 1 below shows that computing the max-flow on $G_{S,k,t}$ is equivalent to solving Problem 2. The main idea is that the flow passing through an interval arc is equivalent to the selection of that arc in the solution S'. The capacity $k - t$ imposed on the backbone arcs not incident to

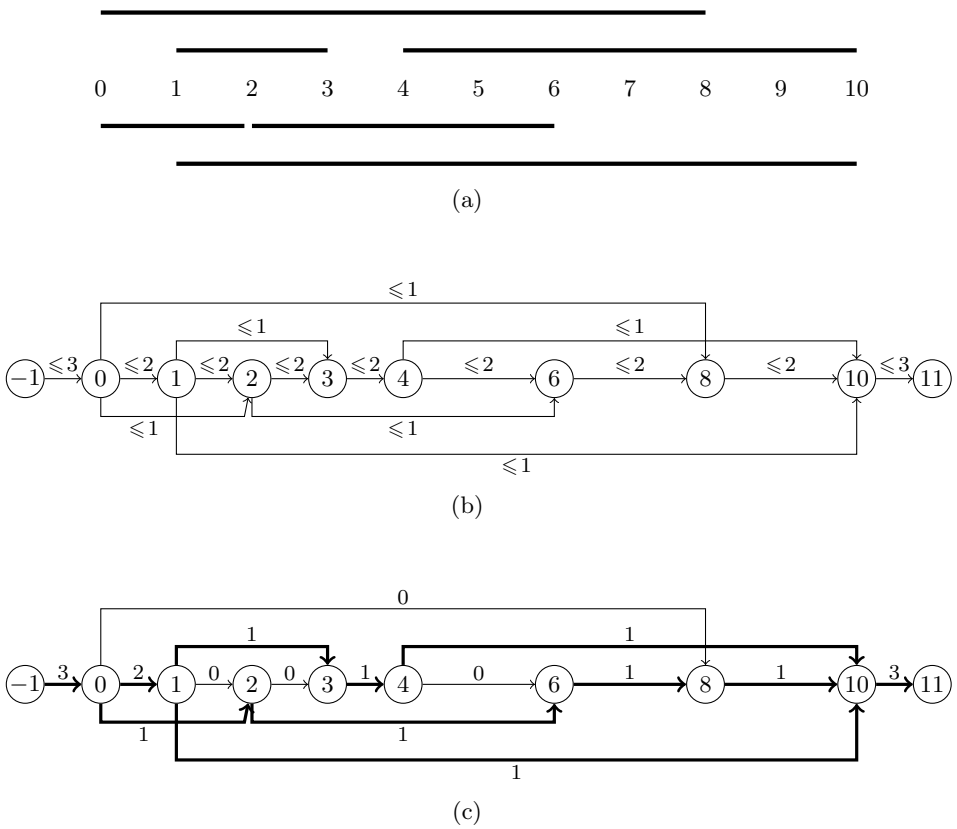

Figure 2: An example for the reduction of Problem 2 to a max flow problem. In Fig. 2(a) an instance (S, k, t) of Problem 2 consisting of 5 intervals, where we assume $k = 3$ and $t = 1$. One solution is obtained by removing the interval $[0, 8)$. In Fig. 2(b) the graph $G_{S,k,t}$ whose arcs are labeled by capacities. In Fig. 2(b) a max-flow of value 3 in $G_{S,k,t}$; the arc labels now indicate their flow value. Arc $(0, 8)$ has flow value 0.

$s-1$ or to $f+1$ implies that for any $p \in [s, f)$ we have at most $k-t$ intervals not covering p, thus at least t intervals covering p.

Theorem 1. *Problem 2 admits a solution on an instance (S, k, t) if and only if the max-flow in $G_{S,k,t}$ has value k, and this solution can be retrieved from any integral max-flow on $G_{S,k,t}$.*

Proof. We prove the forward implication first. Let $S' \subseteq S$ be a solution to an instance (S, k, t) of Problem 2. See also Fig. 2(c) for an example. We construct a flow φ in $G_{S,k,t}$ by first assigning $\varphi(s-1, s) = \varphi(f, f+1) = k$. Since the capacity of these two arcs is k, and since there is no other arc out-going from $s-1$ or in-coming to $f+1$, this will imply that φ is a max-flow in $G_{S,k,t}$. For any interval arc (s_i, f_i), we set $f(s_i, f_i)$ to 1 if and only if the corresponding interval $[s_i, f_i)$ belongs to S'. Finally, let (i, j) be any backbone arc different from $(s-1, s)$ and $(f, f+1)$. Since all $p \in [i, j)$ are covered by the same number of intervals in S', say $t_{i,j}$, and $t_{i,j} \leqslant t$, we set $f(i, j) = k - t_{i,j}$.

So far we have obtained that φ satisfies the capacity constraints on $G_{S,k,t}$. It remains to show that the flow conservation property holds for every vertex other than the source and the sink. Let i be such a vertex, and let (i, j) be its out-going backbone arc, and let (ℓ, i) be its in-coming backbone arc. The value of the flow out-going from i equals $k - t_{i,j}$ plus the number of intervals in S' with i as left extremity. This equals k minus the number of intervals strictly covering i. Similarly, the

value of the flow in-coming to i equals $k - t_{\ell,i}$ plus the number of intervals ending at i. This also equals k minus the number of intervals strictly covering i, thus showing that the flow conservation property holds for i.

For the reverse implication, let φ be an integral max-flow in $G_{S,k,t}$ of value k (such an integral flow exists because all capacities are integers). The solution $S' \subseteq S$ to problem Problem 2 consists of those intervals $[s_i, f_i)$ such that the corresponding interval arc (s_i, f_i) has flow value 1. Let (i, j) be an arbitrary backbone arc in $G_{S,k,t}$ not incident to the source or to the sink. We need to show that all $p \in [i, j)$ are covered by at least t intervals of S' and by at most k intervals of S'. Point p is covered by at most k intervals because f has value k. Point p is covered by at least t intervals because the capacity of the backbone arc (i, j) is $k - t$. □

Observe that $G_{S,k,t}$ has $O(n)$ vertices and arcs. Thus, the specialized max-flow algorithm from [13] applies, leading to the following corollary.

Corollary 2. *Problem 2 is solvable in time $O(n^2/\log n)$ by solving the max-flow problem on $G_{S,k,t}$.*

We can use the above corollary to solve the maximization problem (Problem 1) by a doubling/binary search technique as follows. We apply the corollary for $t = 1$, $t = 2$, $t = 4$, ..., until finding t^+ where for $t^- := 2t^+$ the decision problem does no longer have a solution. Binary search on decision problem instances with $t \in [t^+..t^-]$ gives the minimum coverage $t = \texttt{OPT}$ of the optimal solution to the maximization problem.

Corollary 3. *Problem 1 is solvable in time $O(n^2 \log \texttt{OPT}/\log n)$, where $\texttt{OPT}$, $\texttt{OPT} \leqslant k$, is the minimum coverage of an optimal solution.*

4. Tailored max-flow algorithm

Now we give a tailored max-flow algorithm to our problem to achieve a better running time when $k = o(n/\log n)$, based on the Ford-Fulkerson [7] max-flow algorithm. Recall that this classical textbook algorithm (see e.g. [5, Section 26.2]) finds an augmenting path in the residual network and adjusts the flow network along the same path so as to increase the total flow. When there is no augmenting path left in the residual network, the flow found is maximum. Assuming integral capacities (so that the flow increases at least by one unit each augmentation step), the running time is $O(|E||\varphi^*|)$, where E is the set of arcs of the flow network and $|\varphi^*|$ is the value of the maximum flow.

Now consider running Ford-Fulkerson on an instance resulting from the reduction of Sec. 3. We observe that flow $k - t$ can be sent through the backbone arcs from source to sink. Thus, we can directly initialize the network with a flow of value $k - t$, and start running Ford-Fulkerson from that initial feasible flow. We need at most $t \leqslant k$ augmentation steps each requiring $O(n)$ time, and thus we obtain the following result.

Theorem 4. *Problem 2 is solvable in time $O(nk)$.*

Using again the above doubling/binary search algorithm on the decision problem we solve the maximization problem:

Corollary 5. *Problem 1 is solvable in time $O(nk \log \texttt{OPT})$, where $\texttt{OPT}$, $\texttt{OPT} \leqslant k$, is the minimum coverage of an optimal solution.*

5. An $O(n \log n)$ time approximation algorithm

Since for large k the best complexity we achieve for our interval scheduling problem is still almost quadratic, we also study a way to find approximate solutions: In this section we present an approximation algorithm running in time $O(n \log n)$, with approximation ratio $\frac{k}{\lfloor k/2 \rfloor}$. For k even, this is a 2-approximation algorithm. First, we extend some concepts introduced in Section 2, and then make some preliminary observations. Then we describe the algorithm, and finally show how it can be implemented so that it achieves the stated running time.

Let us extend the definition of minimum coverage to intervals, so that $\mathsf{mincov}([s_i, f_i)) = min_{p \in [s_i, f_i)} \mathsf{cov}(p)$. When an interval has minimum coverage smaller or equal to $\lfloor k/2 \rfloor$ we say that such an interval is *crucial*; otherwise, we call it *expendable*. The following result is the key idea behind the approximation algorithm.

Lemma 1. *Given an input (S, k) to the interval scheduling maximizing minumum coverage problem, and a point p, if $\mathsf{cov}_S(p) = k' > k$, there at least $k' - k$ intervals covering p that are expendable.*

Proof. We proceed by contradiction: Let us assume that there are $k' > k$ intervals that cover p and that all of them are *crucial*. For every *crucial* interval, there is at least one point contained in it that has coverage smaller than or equal to $\lfloor k/2 \rfloor$. Let us call these *supporting points*. It is not possible that p is a supporting point, so those must be either larger or smaller than p. If we separate them among those that are larger than p and those smaller than p, one of those sets must have at least $\lceil \frac{k+1}{2} \rceil$ intervals. Without loss of generality, let us assume that there are at least $\lceil \frac{k+1}{2} \rceil$ intervals that have a supporting point larger than p. Among those, if we inspect the interval that have the smallest supporting point (i.e. the one that is closest to p), this supporting point must be covered by the totality of the $\lceil \frac{k+1}{2} \rceil$ intervals that have the supporting point in the same direction. Therefore the coverage of such supporting point must be equal to or greater than $\lceil \frac{k'}{2} \rceil > \lfloor \frac{k}{2} \rfloor$, which contradicts the fact that such an interval is crucial. □

The algorithm proceeds as follows. The original set of intervals S will be treated as the set of current candidates. For every interval, we need to compute its maximum and minimum coverage. This allows to detect *crucial* intervals, which will never be removed from S. Then, the intervals delimiters are traversed from left to right. Whenever a delimiter p is found to have coverage k' greater than k, then $k' - k$ intervals covering p must be removed from the set of candidates. By Lemma 1, we know that among the k' intervals covering p, there are at least $k' - k$ intervals that are *expendable*, so we can delete those safely. For every removed interval $[s_i, f_i)$, we need to update the coverage of all the delimiters that are contained in $[s_i, f_i)$. It is easy to see that the optimal solution to Problem 1 is bounded by k, and, because the algorithm never removes *crucial* intervals, that the approximation ratio is $\rho = \frac{k}{\lfloor k/2 \rfloor}$.

Now we show the data structures that allow us to run the algorithm in $O(n \log n)$ time. We build a perfect binary search tree with delimiters of the intervals as leaves. Initially, each leaf stores the number of intervals overlapping it, i.e. the coverage of the delimiter. This information can be computed by a sweep from left to right through the delimiters, incrementing a counter on the start of an interval and decrementing the counter on the end of an interval, and storing the intermediate counter values to the leaves. We regard the tree as a one-dimensional range search tree [6, Section 5.1], such that internal nodes store keys to allow search towards the leaves by the delimiter. We annotate the tree with maximum and minimum of leaf coverages inside each subtree. For leaves, the maximum and minimum correspond to their stored coverage values. For internal nodes these

values can be computed bottom-up. In addition, we annotate each node of the tree with a *balance* counter, initially set to 0, to support deletion of intervals, as follows.

The approximation algorithm goes through the intervals in the order of their start points. At each such *query* interval q, we locate a set $V(q)$ of $O(\log n)$ internal nodes that form a partition of the query interval as in [6, Section 5]. The maximum and minimum coverage encountered at the query interval can be computed by taking $\max_{v \in V(q)} v.\texttt{maxcov} + v.\texttt{balance}$ and $\min_{v \in V(q)} v.\texttt{mincov} + v.\texttt{balance}$, respectively, where $v.\texttt{maxcov}$ and $v.\texttt{mincov}$ are the minimum and maximum coverages of the corresponding subtrees, and $v.\texttt{balance}$, mentioned above, stores a value indicating how much each coverage inside the subtree has changed during earlier steps of the algorithm. These obtained maximum and minimum coverage values decide if q is deleted or not. If q is deleted, we need to update the coverages in the tree. This is done by updating $v.\texttt{balance} = v.\texttt{balance} - 1$ for all $v \in V(q)$. We propagate the effect of these decrements up to the root, by recomputing maxima and minima on the affected paths, considering $v.\texttt{maxcov} + v.\texttt{balance}$ and $v.\texttt{mincov} + v.\texttt{balance}$ when computing those values. Finally, to guarantee that all $v.\texttt{balance}$ values are maintained correctly, we need to propagate those values down in the tree when querying an interval: during the location of a delimiter of a query interval, and moving from parent p of v to v, we set $v.\texttt{balance} = v.\texttt{balance} + p.\texttt{balance}$, $w.\texttt{balance} = w.\texttt{balance} + p.\texttt{balance}$, and $p.\texttt{balance} = 0$, where w is the other child of p. Processing each interval takes $O(\log n)$ time, which proves the running time claim. We have thus obtained the following result.

Theorem 6. *There is an $O(n \log n)$ time approximation algorithm to Problem 1 that finds a solution with minimum coverage at least $\frac{\lfloor k/2 \rfloor}{k}\texttt{OPT}$, where $\texttt{OPT}$ is the minimum coverage of an optimal solution.*

Acknowledgements

We wish to thank Chao Xu for bringing to our attention the specialized flow algorithms that are suitable for the inputs resulting from our reduction.

This work was partially supported by the Academy of Finland grants 284598 to VM, VS, and DV, and 274977 to AT.

References

[1] Esther M Arkin and Ellen B Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1–8, 1987.

[2] Khalid I Bouzina and Hamilton Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3-4):379–393, 1996.

[3] Peter Brucker and Lars Nordmann. Thek-track assignment problem. *Computing*, 52(2):97–122, 1994.

[4] Rudi Cilibrasi, Leo Van Iersel, Steven Kelk, and John Tromp. On the complexity of several haplotyping problems. In *Algorithms in bioinformatics*, pages 128–139. Springer, 2005.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[6] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1998. Second edition.

[7] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[8] Udaiprakash I Gupta, Der-Tsai Lee, and JY-T Leung. An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, (11):807–810, 1979.

[9] Antoon W.J. Kolen, Jan Karel Lenstra, Christos H. Papadimitriou, and Frits C.R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[10] Mikhail Y Kovalyov, CT Ng, and TC Edwin Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331–342, 2007.

[11] Ross Lippert, Russell Schwartz, Giuseppe Lancia, and Sorin Istrail. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in bioinformatics*, 3(1):23–31, 2002.

[12] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I Tomescu. *Genome-Scale Algorithm Design*. Cambridge University Press, 2015.

[13] James B. Orlin. Max Flows in $O(nm)$ Time, or Better. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 765–774, New York, NY, USA, 2013. ACM.

[14] Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo van Iersel, Leen Stougie, Gunnar W Klau, and Alexander Schönhuth. Whatshap: Haplotype assembly for future-generation sequencing reads. In *Research in Computational Molecular Biology*, pages 237–249. Springer, 2014.